RCC User's Manual

Version 1.0.12

April 2006

1 Introduction 4
1.1 General 4
1.2 Installation on host platform 4
1.2.1 Host requirements 4
1.2.2 Windows- specific issues 4
1.2.3 Installation 5
1.3 Contents of /opt/rtems-4.6 5
1.4 RCC tools 5
1.5 Documentation 6
1.6 Support 6

2 Using RCC 7
2.1 General development flow 7
2.2 sparc-rtems-gcc options 7
2.3 RTEMS applications 7
2.4 Floating-point considerations 8
2.5 LEON SPARC V8 instructions 8
2.6 Memory organisation 8
2.7 Board-support packages (BSPs) 9
2.8 Making boot-proms 9
2.9 Simple examples 10

3 Execution and debugging 11
3.1 TSIM 11
3.2 GRMON 11
3.3 GDB with GRMON and TSIM 13
3.4 Using DDD graphical front-end to gdb 14

4 sparc-rtems-mkprom manual 15

1 Introduction

1.1 General

This document describes the RTEMS LEON/ERC32 GNU cross-compiler system (RCC). Discussions are provided for the following topics:

- contents and directory structure of RCC
- compiling and linking LEON and ERC32 RTEMS applications
- usage of GRMON and MKPROM
- debugging application with GDB

RCC is a multi-platform development system based on the GNU family of freely available tools with additional 'point' tools developed by Cygnus, OAR and Gaisler Research. RCC consists of the following packages:

- GCC-3.2.3 C/C++ compiler
- GNU binary utilities 2.13.1 with support for LEON UMAC/SMAC instructions
- RTEMS-4.6.5 C/C++ real-time kernel with LEON2, LEON3 and ERC32 support
- Newlib-1.13.1 standalone C-library
- GDB-6.3 SPARC cross-debugger
- DDD graphical front-end for GDB
- mkprom-erc32 boot-prom builder (ERC32)
- sparc-rtems-mkprom boot-prom builder (LEON2/LEON3)

1.2 Installation on host platform

1.2.1 Host requirements

RCC is provided for two host platforms: linux/x86 and MS Windows. The following are the platform system requirements:

Linux:	Linux-2.4.x, glibc-2.3 (or higher)
Windows:	Cygwin-1.1.7 (or higher)

1.2.2 Windows- specific issues

To run on Windows platforms, the Cygwin-1.3.10 unix emulation layer needs to be installed. The directories where the compiler is installed (/opt/rtems-4.6) and where applications are compiled needs both to be mounted in binmode. The graphical front-end for gdb (DDD) is not provided with the cygwin version of RCC. It should be installed during the cygwin installation, together with the cygwin/X package. Once installed, the cygwin version of DDD operates the same way as the linux version.

1.2.3 Installation

The RCC directory tree is compiled to reside in the /opt/rtems-4.6 directory on all platforms. After obtaining the bzipped tarfile with the binary distribution, uncompress and untar it in a suitable location - if this is not /opt/rtems-4.6 then a link have to be created to point to the location of the RCC directory. The distribution can be installed with the following commands:

```
cd /opt

bunzip2 -c sparc-rtems-4.6.5-gcc-3.2.3-1.0.12-linux.tar.bz2 | tar xf -
```

After the compiler is installed, add /opt/rtems-4.6/bin to your executables search path.

1.3 Contents of /opt/rtems-4.6

The created rtems directory has the following sub-directories:

```
bin                Executables
doc                RCC documentation
include            Host includes
lib                Host libraries
man                Main pages for GNU tools
sparc-rtems        Sparc target libraries
src                Various sources
```

1.4 RCC tools

The following tools are included in RCC:

```
ddd                Graphic X11 front-end to GDB
grmon              LEON combined remote monitor and simulator
mkprom-erc32       ERC32 boot-prom builder
sparc-rtems-mkprom LEON2/3 boot-prom builder
protoize           GNU protoize utility
sparc-rtems-ar     Library archiver
sparc-rtems-as     Cross-assembler
sparc-rtems-c++    C++ cross-compiler
sparc-rtems-c++filt Utility to demangle C++ symbols
sparc-rtems-g++    Same as sparc-rtems-c++
sparc-rtems-gasp   Assembler pre-processor
sparc-rtems-gcc    C/C++ cross-compiler
sparc-rtems-gdb    Debugger
sparc-rtems-ld     GNU linker
sparc-rtems-nm     Utility to print symbol table
sparc-rtems-objcopy Utility to convert between binary formats
sparc-rtems-objdump Utility to dump various parts of executables
sparc-rtems-ranlib Library sorter
sparc-rtems-size   Utility to display segment sizes
sparc-rtems-strings Utility to dump strings from executables
sparc-rtems-strip  Utility to remove symbol table
unprotoize GNU     Unprotoize utility
```

1.5 Documentation

The RCC documentation is distributed separately in two packages: RTEMS manuals and GNU tools manuals.

GNU manuals:

```
as.pdf            Using as - the GNU assembler
binutils.pdf      The GNU binary utilities
cpp.pdf           The C Preprocessor
ddd.pdf           DDD - The Data Display Debugger
gcc.pdf           Using and porting GCC
gdb.pdf           Debugging with GDB
```

RTEMS manuals:

```
FAQ.pdf           RTEMS Frequently Asked Questions
bsp_howto.pdf     BSP and Device Driver Development Guide
c_user.pdf        RTEMS C User's Guide (this is the one you want!)
develenv.pdf      RTEMS Development environment guide
filesystem.pdf    RTEMS Filesystem Design Guide
itron.pdf         RTEMS ITRON 3.0 User's Guide
networking.pdf    RTEMS Network Supplement
new_chapters.pdf  RTEMS Newly added features
porting.pdf       RTEMS Porting Guide
posix1003-1.pdf   RTEMS POSIX 1003.1 Compliance Guide
posix.pdf         RTEMS POSIX API User's Guide
relnotes.pdf      RTEMS Release Notes
sparc.pdf         RTEMS SPARC Applications Supplement
start.pdf         Getting Started with RTEMS for C/C++ Users
```

The documents are all provided in PDF format, with searchable indexes.

1.6 Support

The RCC compiler system is provided freely without any warranties. Technical support can be obtained from Gaisler Research through the purchase of a technical support contract. See www.gaisler.com for more details.

2 Using RCC

2.1 General development flow

Compilation and debugging of applications is typically done in the following steps:

1. Compile and link program with gcc
2. Debug program using a simulator (gdb connected to grmon using simulator backend)
3. Debug program on remote target (gdb connected to grmon using monitor backend)
4. Create boot-prom for a standalone application

RCC supports multi-tasking real-time C/C++ programs based on the RTEMS kernel. Compiling and linking is done in much the same manner as with a host-based gcc.

2.2 sparc-rtems-gcc options

The gcc compiler has been modified to support the following addtional options:

-qleon2	generate LEON2 executable.
-tsc691	generate ERC32 executable.

Other usefull (standard) optionas are:

-g	generate debugging information - must be used for debugging with gdb
-msoft-float	emulate floating-point - must be used if no FPU exists in the system
-mv8	generate SPARC V8 mul/div instructions- only LEON with hardware multiply and divide configured
-O2	optimize code - should be used for optimum performance and minimal code size

Other gcc options are explained in the gcc manual (gcc.pdf).

2.3 RTEMS applications

To compile and link an RTEMS application, use 'sparc-rtems-gcc':

```
sparc-rtems-gcc -g -O2 rtems-hello.c -o rtems-hello
```

RCC creates executables for LEON3 by default. To generate executables for LEON2 or ERC32 add -qleon2 or -tsc691 switches during both compile and link stages. The default load address is start of RAM, i.e. 0x40000000 for LEON2/3 and 0x2000000 for ERC32. Other load addresses can be specified through the use of the -Ttext option (see gcc manual).

RCC uses the sources of RTEMS-4.6.5 with minor patches, and allows re-compilation if a modification has been made to a bsp or the kernel. Install the RTEMS sources in /opt/rtems-4.6/src, and re-compile and install with:

```
cd /opt/rtems-4.6/src
make install
```

2.4 Floating-point considerations

If the targeted LEON2/3 (or ERC32) processor has no floating-point hardware, then all applications must be compiled (and linked) with the -msoft-float option to enable floating-point emulation. When running the program on the TSIM simulator or GRMON with simulator backend, the simulator should be started with the -nfp option (no floating-point) to disable the FPU.

2.5 LEON SPARC V8 instructions

Both LEON2 and LEON3 processors can be configured to implement the SPARC V8 multiply and divide instructions. The RCC compiler does by default NOT issue those instructions, but emulates them trough a library. To enable generation of mul/div instruction, use the -mv8 switch during both compilation and linking. The -mv8 switch improves performance on compute-intensive applications and floating-point emulation.

LEON2/3 also supports multiply and accumulate (MAC). The compiler will never issue those instructions, they have to be coded in assembly. Note that the RCC assembler and other utilities are based on a modified version of GNU binutils-2.11 that supports the LEON MAC instructions.

2.6 Memory organisation

The resulting RTEMS executables are in elf format and has three main segments; text, data and bss. The text segment is by default at address 0x40000000 for LEON2/3 and 0x2000000 for ERC32, followed immediately by the data and bss segments. The stack starts at top-of-ram and extends downwards.

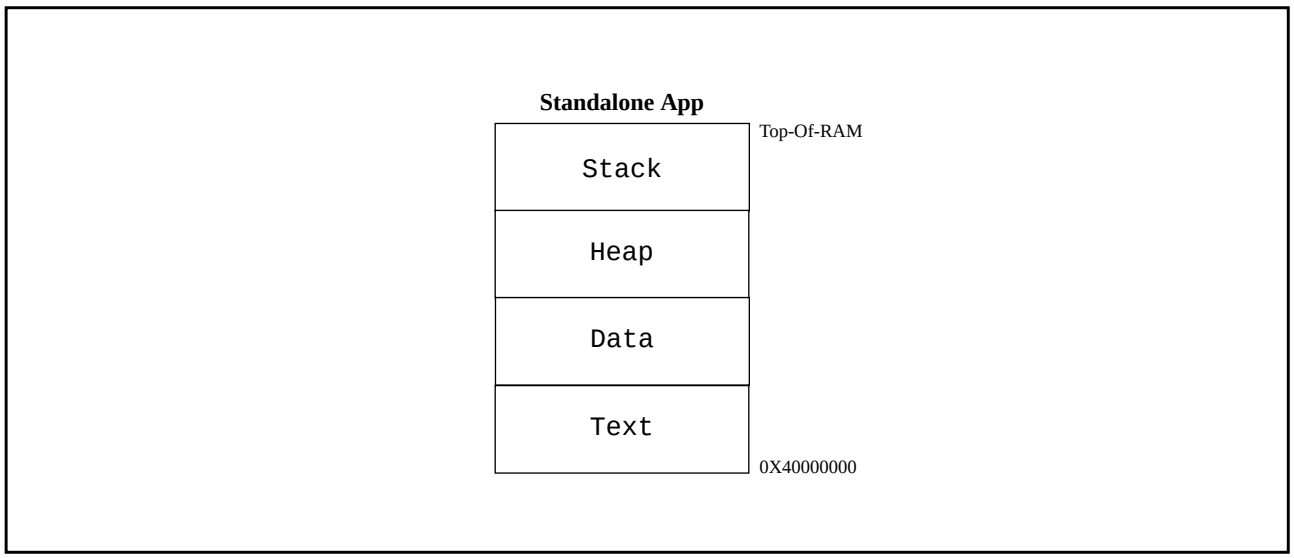

Figure 1: RCC RAM applications memory map

The SPARC trap table always occupies the first 4 Kbyte of the .text segment.

2.7 Board-support packages (BSPs)

RCC includes board-support packages for LEON2, LEON3 and ERC32. BSPs provide interface between RTEMS and target hardware through initialization code specific to target processor and a number of device drivers. Console and timer drivers are supported for all three processors.

The LEON2/3 BSPs support three network drivers: the Gaisler Research GRETH MAC, OpenCores Ethernet MAC and the LAN91C111. The default driver is the GRETH MAC. To select a different network driver, one of following defines should be set before including the bsp.h file:

```
#define RTEMS_BSP_NETWORK_DRIVER_ATTACH RTEMS_BSP_NETWORK_DRIVER_ATTACH_OPENETH

#define RTEMS_BSP_NETWORK_DRIVER_ATTACH RTEMS_BSP_NETWORK_DRIVER_ATTACH_SMC91111
```

See src/examples/samples/rtems_ttcp.c or rtems_http.c for a sample networking applications.

LEON2 and ERC32 BSPs assume a default system resource configuration such as memory mapping of on-chip devices and usage of interrupt resources. LEON3 systems are based on GRLIB plug&play configuration, and are thereby highly configurable regarding memory mapping and interrupt routing. At start-up, LEON3 BSP scans the system bus to obtain system configuration information. Device drivers support a number of devices which are automatically recognized, initiated and handled by the device drivers.

The console driver supports one to eight GR APB UARTs. The first UART is registered under name "/dev/console", second and third UARTs get names "/dev/console_b" and "dev/console_c" and so on. LEON3 BSP requires at least one GR APB UART.

The timer driver uses GR General Purpose Timer (GPT). The driver handles GPT timer 0 and the lowest interrupt request line used by GPT. GPT timer 0 and lowest request line should never be used by an RTEMS application. If application needs to use more timers GPT should be configured to have two or more timers using separate request lines. Timer 0 interrupt can not be shared with other devices or GPT timers 2-7.

For more information on how to configure a system based on GRLIB see GRLIB IP Library User's Manual.

2.8 Making boot-proms

RTEMS applications are linked to run from beginning of RAM. To make a boot-PROM that will run from the PROM on a standalone target, use the sparc-rtems-mkprom utility. This will create a compressed boot image that will load the application into RAM, initiate various processor registers, and finally start the application. Sparc-rtems-mkprom will set all target dependent parameters, such as memory sizes, waitstates, baudrate, and system clock. The applications do not set these parameters themselves, and thus do not need to be re-linked for different board architectures.

The example below creates a LEON3 boot-prom for a system with 1 Mbyte RAM, one waitstate during write, 3 waitstates for rom access, and 40 MHz system clock. For more details see the mkprom manual

```
sparc-rtems-mkprom -ramsz 1024 -ramwws 1 -romws 3 hello.exe -freq 40 hello.exe
```

Note that sparc-rtems-mkprom creates binaries for LEON2/3. To generate binaries for ERC32, use mkprom-erc32. To create an SRECORD file for a prom programmer, use objcopy:

```
sparc-rtems-objcopy -O srec rtems-hello rtems-hello.srec
```

2.9 Simple examples

Following example compiles the famous "hello world" program and creates a boot-prom in SRECORD format:

```
bash-2.04$ sparc-rtems-gcc -mv8 -msoft-float -O2 rtems-hello.c -o rtems-hello

bash-2.04$ sparc-rtems-mkprom -freq 40 -dump -baud 38400 -ramsize 1024 -rmw rtems-hello
bash-2.04$ sparc-rtems-objcopy -O srec rtems-hello rtems-hello.srec
bash-2.04$
```

Several example C programs can be found in /opt/rtems-4.6/src/examples/samples.

3 Execution and debugging

Applications built by RCC can be debugged on the TSIM LEON/ERC32 simulator, or on target hardware using the GRMON debug monitor (LEON only). Both TSIM and GRMON can be conncted to the GNU debugger (gdb) for full source-level debugging.

3.1 TSIM

The TSIM simulator can emulate a full ERC32 and LEON2/3 system with on-chip peripherals and external memories. For full details on how to use TSIM, see the TSIM User's Manual. Below is a simple example that shows how the 'hello world' program is run in the simulator:

```
$ tsim-leon3 rtems-hello

 TSIM/LEON3 SPARC simulator, version 2.0.4a (professional version)

 Copyright (C) 2001, Gaisler Research - all rights reserved.
 For latest updates, go to http://www.gaisler.com/
 Comments or bug-reports to tsim@gaisler.com

using 64-bit time
serial port A on stdin/stdout
allocated 4096 K RAM memory, in 1 bank(s)
allocated 2048 K ROM memory
icache: 1 * 4 kbytes, 16 bytes/line (4 kbytes total)
dcache: 1 * 4 kbytes, 16 bytes/line (4 kbytes total)
section: .text, addr: 0x40000000, size 92096 bytes
section: .data, addr: 0x400167c0, size 2752 bytes
read 463 symbols
tsim> go
resuming at 0x40000000
Hello World


Program exited normally.
tsim>
```

3.2 GRMON

GRMON is used to download, run and debug LEON2/3 software on target hardware. For full details on how to use GRMON, see the GRMON User's Manual. Below is a simple example that shows how the 'hello world' program is downloaded and run:

```
$ grmon -u -jtag

 GRMON LEON debug monitor v1.1.11

 Copyright (C) 2004,2005 Gaisler Research - all rights reserved.
 For latest updates, go to http://www.gaisler.com/
 Comments or bug-reports to support@gaisler.com

 using JTAG cable on parallel port
 JTAG chain: xc3s1500 xcf04s xcf04s

 initialising ............
 detected frequency:  41 MHz
 GRLIB build version: 1347
```

```
 Component                            Vendor
 LEON3 SPARC V8 Processor             Gaisler Research
 AHB Debug UART                       Gaisler Research
 AHB Debug JTAG TAP                   Gaisler Research
 GR Ethernet MAC                      Gaisler Research
 LEON2 Memory Controller              European Space Agency
 AHB/APB Bridge                       Gaisler Research
 LEON3 Debug Support Unit             Gaisler Research
 Nuhorizons Spartan3 I/O interfac     Gaisler Research
 OC CAN controller                    Gaisler Research
 Generic APB UART                     Gaisler Research
 Multi-processor Interrupt Ctrl       Gaisler Research
 Modular Timer Unit                   Gaisler Research

 Use command 'info sys' to print a detailed report of attached cores

grlib> lo rtems-hello
section: .text at 0x40000000, size 92096 bytes
section: .data at 0x400167c0, size 2752 bytes
total size: 94848 bytes (339.7 kbit/s)
read 463 symbols
entry point: 0x40000000
grlib> run
Hello World
grlib>
```

Note that the program was started from address 0x40000000, the default start address.

GRMON can also be used to program the boot-PROM image created by sparc-rtems-mkprom into the target's flash PROM.

```
grmon[grlib]> flash unlock all
grmon[grlib]> flash erase all
Erase in progress
Block @ 0x00000000 : code = 0x00800080  OK
Block @ 0x00004000 : code = 0x00800080  OK
                ...
grmon[grlib]> flash load prom.out
section: .text at 0x0, size 54272 bytes
total size: 54272 bytes (93.2 kbit/s)
read 45 symbols
grmon[grlib]> flash lock all
```

When boot-PROM is run (i.e. after reset) it will initialize various LEON registers, unpack the application to the RAM and start the application. The output on console when running "hello world" from PROM is shown below:

```
 MkProm LEON3 boot loader v1.2
 Copyright Gaisler Research - all right reserved

 system clock    : 40.0 MHz
 baud rate       : 38352 baud
```

```
  prom           : 512 K, (2/2) ws (r/w)
  sram           : 1024 K, 1 bank(s), 0/0 ws (r/w)

  decompressing .text
  decompressing .data

  starting rtems-hello

Hello World
```

The application must be re-loaded with the 'load' command before it is possible to re-execute it.

3.3 GDB with GRMON and TSIM

To perform source-level debugging with gdb, start TSIM or GRMON with -gdb or enter the 'gdb'command at the prompt. Then, attach gdb by giving command 'tar extended-remote localhost:2222' to gdb when connecting to GRMON or 'tar extended-remote localhost:1234' when connecting to TSIM. Note that RTEMS applications do not have a user-defined main() function as ordinary C-programs. Instead, put a breakpoint on Init(), which is the default user-defined start-up function.

```
jupiter> sparc-rtems-gdb rtems-hello
GNU gdb 6.3
Copyright 2002 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "--host=i686-pc-linux-gnu --target=sparc-tsim-elf"...
(gdb) tar extended-remote localhost:2222
Remote debugging using localhost:2222
(gdb) load
Loading section .text, size 0x164e0 lma 0x40000000
Loading section .jcr, size 0x4 lma 0x400164e0
Loading section .data, size 0xaa8 lma 0x400164e8
Start address 0x40000000, load size 94092
Transfer rate: 57902 bits/sec, 277 bytes/write.
(gdb) break Init
Breakpoint 2 at 0x400011f8: file rtems-hello.c, line 33.
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y

Starting program: /opt/rtems-4.6/src/examples/samples/rtems-hello

Breakpoint 2, Init (ignored=0) at rtems-hello.c:33
33        printf("Hello World\n");
(gdb) cont
Continuing.
Hello World
Program exited with code 0363.
```

The application must be re-loaded with the 'load' command before it is possible to re-execute it.

3.4 Using DDD graphical front-end to gdb

DDD is a graphical front-end to gdb, and can be used regardless of target. To start DDD with the sparc-rtems-gdb debugger use:

```
ddd  --debugger sparc-rtems-gdb
```

You might need the full path in front of DDD if you already have a version of ddd installed. The required gdb commands to connect to a target can be entered in the command window. See the GDB and DDD manuals for how to set the default settings. If you have problems with getting DDD to run, run it with --check-configuration to probe for necessary libraries etc. DDD has many advanced features, see the on-line manual under the 'Help'menu.

On windows/cygwin hosts, DDD must be started from an xterm shell. First launch the cygwin X-server by issuing 'startx' in a cygwin shell, and the launch DDD in the newly created xterm shell.

4 sparc-rtems-mkprom manual

NAME

sparc-rtems-mkprom

SYNOPSYS

sparc-rtems-mkprom [*options*] input_files

DESCRIPTION

The mkprom utility program creates boot-images for programs compiled with RCC. It encapsulates the application in a loader suitable to be placed in a boot prom. The application is compressed with a modified LZSS algorithm, typically achieving a compression factor of 2. The loader initiates the system according to the specified parameters. The loader operates in the following steps:

- The register files of IU and FPU (if present) are initialised.
- The LEON control, waitstate and memory configuration registers are set according to the specified options.
- The ram is initialised and the application is decompressed and installed.
- The text part of the application is optionally write-protected, except the lower 4K where the traptable is assumed to reside.
- Finally, the application is started, setting the stack pointer to the top of ram.

sparc-elf-mkprom will perform initialization of a LEON2 or LEON3 system with a standard configuration. Initialization code for additional peripherals can be provided in *bdinit1* and *bdinit2* functions (see *-bdinit* switch under General Options). If the LEON system is configured without some of standard configuration peripherals, sparc-rtems-mkprom should be called with *-noinit* and *-bdinit* switches and with system specific initialization code provided in bdinit.o.

GENERAL OPTIONS

-baud *baudrate*

Set console UART baudrate to *baudrate*. Default value is 19200.

-bdinit

The user can optionally call two user-defined routines, bdinit1() and bdinit2(), during the boot process. bdinit1() is called after the LEON registers have been initialised but before the memory has been cleared. bdinit2() is called after the memory has been initialised but before the application is loaded. Note that when bdinit1() is called, the

stack has not been setup meaning that bdinit1() must be a leaf-routine and not allocate any stack space (no local variables). When -bdinit is used, a file called bdinit.o must exist in the current directory, containing the two routines.

-dump

The intermediate assembly code with the compressed application and the LEON register values is put in `dump.s` (only for debugging of `mkprom3`).

-duart *addr*

GR AHB UART (DSU UART) address. Default is 0x80000700.

-**freq** *system_clock*

Defines the system clock in MHz. This value is used to calculate the divider value for the baud rate generator and the real-time clock. Default is 25 MHz.

-**gpt** *addr*

GR General Purpose Timer (GPT) address. Default is 0x80000300.

-**memc** *addr*

ESA's LEON Memory Controller address. Default is 0x80000000.

-**noinit**

Suppress all code which initializes on-chip peripherals such as uarts, timers and memory controllers. This option requires -bdinit to add custom initialisation code, or the boot process will fail.

-**nocomp**

Don't compress application. Decreases loading time on the expense of rom size.

- **o** *outfile*

Put the resulting image in *outfile*, rather then prom.out (default).

-stack *addr*

Sets the initial stack pointer to *addr*. If not specified, the stack starts at top-of-ram.

-uart *addr*

GR APB UART address. Default is 0x80000100.

-v

Be verbose; reports compression statistics and compile commands

input_files

The input files must be in aout or elf32 format. If more than one file is specified, all files are loaded by the loader and control is transferred to the first segment of the first file.

ESA LEON MEMORY CONTROLLER OPTIONS

Following options are used to initiate ESA's LEON memory controller. If other memory controller is used initialization code has to be provided in *bdinit1* or *bdinit2* functions.

-cas *delay*

Set the SDRAM CAS delay. Allowed values are 2 and 3, 2 is default.

-col *bits*
Set the number of SDRAM column bits. Allowed values are 8 - 11, 9 is default.

-nosram
Disables the static RAM and maps SDRAM at address 0x40000000.

-**ramsize** *size*
Defi nesthe **total** available RAM. Used to initialize the in the memory confi guration register(s). The default value is 2048 (2 Mbyte).

-**ramcs** *chip_selects*
Set the number of ram banks to *chip_selects*. Default is 1.

-ramws *ws*
Set the number of waitstates during ram reads **and** writes to *ws*. Default is 0.

-ramrws *ws*
Set the number of waitstates during ram **reads** to *ws*. Default is 0.

-ramwws *ws*
Set the number of waitstates during ram **writes** to *ws*. Default is 0.

-ramwidth *width*
Set the data bus width to 8, 16 or 32-bits, default is 32. The prom width is set through the PIO[1:0] ports.

-refresh *delay*
Set the SDRAM refresh period (in us). Default is 7.8 us, although many SDRAMS actually use 15.6 us.

-rmw Perform read-modify-write cycles during byte and halfword writes.

-**romsize** *size*
Set the rom size to *size*. Default is 512 (512 Kbyte)

-**romws** *ws*
Set the number of rom waitstates during read **and** write to *ws*. Default is 2.

-**romrws** *ws*
Set the number of rom waitstates during **read** to *ws*. Default is 2.

-**romwws** *ws*
Set the number of rom waitstates during **write** to *ws*. Default is 2.

-sdram *size*
The amount of attached SDRAM in Mbyte. 0 by default

-sdrambanks *num_banks*
Set the number of populated SDRAM banks (default is 1).

-trfc *delay*
Set the SDRAM t_{RFC} parameter (in ns). Default is 66 ns.

-trp *delay*
Set the SDRAM t_{RP} parameter (in ns). Default is 20 ns.